Globule: a Platform for Self-Replicating Web Documents

Pierre, G.; van Steen, M.

2001

Globule: a Platform for Self-Replicating Web Documents

Guillaume Pierre
Maarten van Steen
Vrije Universiteit, Amsterdam

Internal report IR-483
January 2001

Abstract

Replicating Web documents at a worldwide scale can help reducing user-perceived latency and wide-area network traffic. This paper presents the preliminary design of Globule, a platform which automates all aspects of such replication: server-to-server peering negotiation, creation and destruction of replicas, selection of the most appropriate replication strategies on a per-document basis, consistency management and transparent redirection of clients to replicas. To facilitate the transition from a non-replicated server to a replicated one, we designed Globule as a module for the Apache Web server. Therefore, converting normal Web documents should require no more than compiling a new module into Apache and editing a configuration file.

vrije Universiteit

Faculty of Mathematics and Computer Science

1 Introduction

Large-scale distributed systems often address performance issues by way of caching and replication. In the Web, attention has traditionally concentrated on caching and in much lesser extent to replication. Recently, Web hosting services have started to emerge as the solution to achieve scalability through replication. In this approach, content is replicated to places where user demand is high. Web hosting services have the advantage over static mirroring of Web sites in that decisions on *where* to replicate content can be made automatically. However, current hosting services apply a single system-wide policy concerning which consistency is needed and how it is to be realized (see, e.g., [2]).

Many protocols have been proposed to achieve caching or replication, each of which presents specific advantages and drawbacks. We have shown in a previous article that no single policy is efficient for replicating all documents. Significant performance improvements can be obtained by replicating each document with the policy that is most suited to its access pattern [6].

To associate a replication policy with each document separately, we consider a document and all its replicas as a single replicated distributed object. All replication issues are encapsulated *inside* the object. Such an architecture allows a document to monitor its own access pattern and to dynamically select the replication policy that suits it best. When a change is detected in access patterns, it can re-evaluate its choice and switch policies on the fly [5].

Although we have demonstrated the feasibility of this approach, we have not yet addressed its integration in the existing Web infrastructure. There are two problems that need to be addressed. First, servers need to be adapted so that they can support per-document policies. Second, servers need to cooperate to allow replicas to be dynamically installed and removed, and to redirect clients to the nearest replica. One additional requirement is that adaptations should fit into the current Web infrastructure, requiring minimal modifications to existing servers.

This paper presents the preliminary design of Globule, a platform for hosting adaptive Web documents. It is designed as a module for the popular Apache server [1]. Converting normal Web documents should require no more than compiling a new module into Apache and editing a configuration file. Globule handles all management tasks: discovering and negotiating with remote sites for hosting replicas; replicating static and some dynamic documents; and transparently redirecting clients to their closest replica.

The paper is structured as follows: Section 2 describes our document and server models; Section 3 details the architecture of the system. Finally, Section 4 shows how such a system can be implemented as an Apache module, and Section 5 concludes.

2 General Model

Our system is made of servers that cooperate in order to replicate Web documents. This section describes our document and server models.

2.1 Document Model

In contrast to most Web servers, we consider a Web document and its replicas as a single distributed entity, and not as a collection of files. Each distributed document is implemented as a single distributed object replicated across the Internet. Each such object implements a standard interface (`put()`, `get()`, `...`) to allow for reading and modification of its content.

Our system is based on Globe, a platform for large-scale distributed objects [8]. Its main novelty is the encapsulation of issues related to distribution and replication *inside* the objects. We designed documents which dynamically select their own replication policy. Our "documents-are-objects" model is also a key to replicating dynamic Web documents.

2.1.1 Adaptive Replicated Web Documents

We have shown in previous papers that significant performance improvements can be obtained over traditional replicated servers by associating each document with the replication strategy that suits it best [5, 6]. Such per-document replication policies are made possible by the encapsulation of replication issues inside each document.

The selection of the best replication policy is realized internally to each document by way of trace-based simulations. Replicas transmit logs of the requests they received to their master site. When significant access pattern modification is detected, the master re-evaluates its choice of replication strategy. To do so, it extracts the most recent trace records and simulates the behavior of a number of replication policies with this trace. Each simulation outputs performance metrics such as client retrieval time, network traffic and consistency. The "best" policy is chosen from these performance figures using a cost function.

2.1.2 Replicating Dynamic Web Documents

Many documents are not made from static content, but are generated on the fly. For each request, a Web server executes a request-specific piece of code whose output is delivered to the client. This code can access external resources such as databases, execute shell commands, and issue network requests for generating a *view* of the dynamic document.

Replicating dynamic documents requires replicating the code as well as all data necessary for its execution. Since our object model does not differentiate between static and dynamic documents, dynamic documents are considered as objects implementing the same interface as other documents. They distinguish from other documents only by their implementation: whereas static documents use always the same implementation to access various internal states, dynamic documents differ in both their internal states and method implementations.

The real issue with respect to dynamic documents arises when trying to convert existing dynamic documents to objects. The Web server must be able to determine automatically which resources are accessed by each dynamic document in order to include them inside the object. This can be difficult for documents such as CGIs, where the server delegates request handling to an arbitrary external program. However, a large portion of dynamic documents such as PHPs and ASPs are in fact scripts interpreted by the Web server itself. In this case, the server knows the semantics of the document, and can often automatically detect which resources are required by the document.

Dynamic documents that the server can not analyze cannot be encapsulated into objects. Therefore, they are not replicated in our approach.

2.2 Cooperative Servers

One important issue for replicating Web documents is to gain access to computing resources in several locations worldwide (CPU, disk space, memory, bandwidth, etc.). On the other hand, adding extra resources locally is easy. Therefore, the idea is to trade cheap local resources for valuable remote ones. Servers automatically negotiate for resource peering. The result of such a negotiation is for a "hosting server" to agree to allocate a given amount of its local resources to host replicas from a "hosted server." The hosted server keeps control on the resources it has acquired: it controls which of its clients are redirected to the hosting server, which documents are replicated there and which replication policies are being used.

Of course, servers may play both "hosting server" and "hosted server" roles at the same time: a server may host replicas from another server, and replicate its own content to a third one. We use these terms only to distinguish roles within a given cooperation session.

By allowing hosted servers to use their local resources, hosting servers will gain the ability of deploying replicas of their own content. The details of this negotiation still have to be figured out.

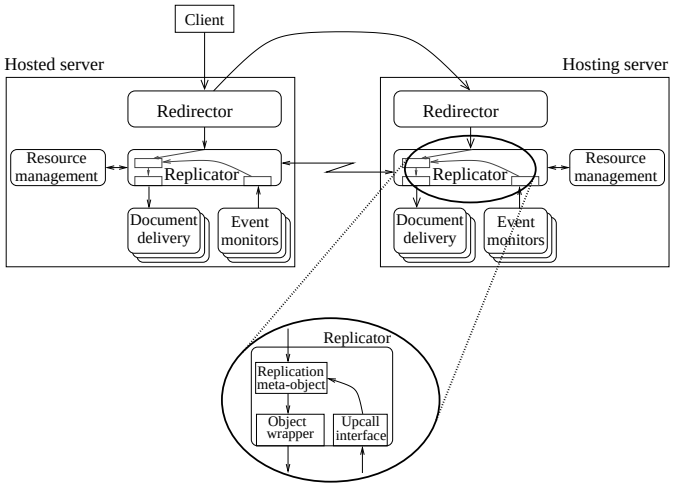

Figure 1: General Architecture

3 System Architecture

Figure 1 shows Globule's architecture. It provides several distinct features: negotiating with remote servers for resource allocation and resource management, document replication and consistency, and automatic client redirection.

3.1 Delegated Resource Management

When a server notices that replicating its documents would improve the quality of service for its clients, it identifies one or more locations where replicas would be needed. Those locations are determined, for example, as the Autonomous Systems from which most of the requests originate.[1] It then locates suitable hosting servers in those locations and negotiates resource allocation.

When a hosted server has acquired appropriate resources on a hosting server, it can use them to create replicas, and redirect clients to them. Its only obligation is to not exceed the amount of resources that have been allocated. This rule is enforced by the hosting server: to prevent excessive use of storage space, a replacement module similar to those of caches is associated to the set of replicas from this hosted server. If a hosted server tries to allocate more resources than allowed, the hosting server will automatically delete other replicas from the same server. Simi-

[1] Usually, a small number of origin Autonomous Systems account for a large fraction of the requests, as shown for example in [6].

lar mechanisms can be setup to enforce limitations in resources such as bandwidth usage [3].

A hosting server having allocated resources for several different hosted servers manages each resource pool separately. Therefore, it must have several replacement module instances, each of which enforces resource limitations to one of the hosted servers. This mechanism is similar to those of partitioned caches [4].

Even though resource limitation is enforced by the hosting server, the hosted server remains in control of its allocated resources. It does so by attaching priority flags to its documents, indicating how important each replica is. When making replacement decisions, the hosting server will take these flags into account in addition to standard parameters such as the frequency of requests and replica size. The more important a replica, the less likely it is to be removed.

3.2 Document Replication

As is also shown in Figure 1, a replica is made of two separate local objects: a document's state which is available in the form of delivery components capable of producing documents, and a replication meta-object which is responsible for enforcing the document's replication policy. Each object can either reside in memory or be marshaled to disk. State transitions between memory and disk are dictated by the per-hosted-server resource limitation module described in Section 3.1.

All replication meta-objects implement a standard interface, but they can have various implementations depending on the replication policy they represent. Each time a request is issued to a document, the server first invokes the document's meta-object. Depending on its policy and internal state, the meta-object decides how to process the request. It can for example decide to reply immediately based on the local replica, or require to first check for freshness, etc.

Once the replication meta-object has authorized for it, the Web server uses one of its standard document delivery modules to answer the request. These can be modules that deliver static documents, or modules that generate a document on request.

Certain replication policies require taking actions at other times than request time, such as periodically checking for a document's freshness, sending invalidations to replicas when the master copy is updated, and processing incoming invalidations. To do so, meta-objects can register to local services for being invoked when certain events take place. For example, a meta-object can request to be woken up periodically or when a given file is updated.

3.3 Client Redirection

Each document is assumed to have a home server. This server is responsible for automatically redirecting clients to their most suitable replica. Knowing the location of clients and replicas, such a selection can be reduced to a shortest path problem.

Two mechanisms can be used to effectively redirect clients to replicas:

- HTTP redirection: when receiving an HTTP request, the server sends a redirection response, indicating from which URL the document should be retrieved. This scheme is very simple, but it is not transparent. That is, browsers display the URL of the mirror site instead of the home site. This may become a problem if, for example, a user bookmarks a mirror page. Later on, when he tries to access the page again, this mirror may have been removed.

- DNS redirection: before accessing a page, a browser needs to request the DNS to convert the server's name into an IP address. After locating an authoritative server for the given domain, the client's DNS server contacts it for actually resolving the name. DNS redirection requires the authoritative server to send customized responses depending on the location of the client [7]. Small TTL values are associated to responses, so that client DNS caches are updated often. A customized authoritative DNS server is necessary, but no other DNS server needs to be modified. This method is fully transparent to the user, since URLs do not need to be modified. On the other hand, it has a coarse granularity: it is not possible to replicate only part of a server, since all requests to this server will be sent to the mirrors.

We decided to use DNS redirection, as did most of the content distribution networks [2]. However, our system's architecture does not depend on this decision; we may later decide to use another mechanism, for example when the HTTP standard will feature more powerful redirection mechanisms.

4 Integration into the Apache Server

In order to allow for easy deployment, we decided to develop Globule as an Apache module. This way, turning a normal Apache server into a replication-aware server would require only compiling a new module into the server and editing a configuration file.

The Apache Web server is built from a modular design, which enables one to easily add new features. It decomposes the treatment for each request into several

steps, such as access checking, actually sending a response back to the client, and logging the request.

Modules can register handler functions to participate in one or more of these steps. When a request is received, the server runs the registered handlers for each step. Modules can then accept or refuse to process the operation; the server tries all the handlers registered for each step until one accepts to process it.

Many third party modules have been developed to extend Apache in a number of ways. Among the most popular ones are server scripting languages such as PHP.

The architecture of Apache provides us all the tools necessary to implement Globule: we can, for example intercept a request before being served by the standard document delivery modules to let the replication meta-objects check for consistency. Servers can communicate by sending HTTP requests to each other.

Although Apache is an HTTP server, one can write modules to implement other protocols. So, for example, one could turn Apache into a DNS server. We plan to use this feature for redirecting clients to mirrors.

5 Conclusion

We have presented Globule, a platform for Web document replication. Globule integrates all necessary services into a single tool: dynamic creation and removal of replicas, consistency management, and automatic client redirection. Globule will be implemented as a module for the Apache server.

The architecture presented in this article is still work in progress, but we hope to release a first prototype soon. Two problems have been left for future work. First, security: the administrator of a server would like to make sure that remote servers which accepted to host his replicas will do it without modifying documents, for example. We plan to use a trust model to solve this. Second, details of server-to-server negotiation still have to be figured out.

When the Globule project is completed, we expect to provide a free cooperative platform for Web document replication that will match the ever-increasing quality of service expectations that users have.

References

[1] *Apache Web server*, `http://www.apache.org/httpd/`.

[2] F. Thomson Leighton and Daniel M. Lewis, *Global hosting system*, United States Patent, Number US6108703, August 2000.

[3] *mod_bandwidth – bandwidth management module for Apache webserver*, `http://www.cohprog.com/mod_bandwidth.html`.

[4] Cristina Duarte Murta, Virgílio Almeida, and Wagner Meira, Jr, *Analyzing performance of partitioned caches for the WWW*, Proceedings of the 3rd International WWW Caching Workshop, June 1998.

[5] Guillaume Pierre, Ihor Kuz, and Maarten van Steen, *Adaptive replicated Web documents*, Technical Report IR-477, Vrije Universiteit, Amsterdam, September 2000, `http://www.cs.vu.nl/~gpierre/publi/ARWD_ir477.html`.

[6] Guillaume Pierre, Ihor Kuz, Maarten van Steen, and Andrew S. Tanenbaum, *Differentiated strategies for replicating Web documents*, Proceedings of the 5th International Web Caching and Content Delivery Workshop, May 2000, `http://www.cs.vu.nl/~gpierre/publi/DSRWD_wcw2000.html`.

[7] Wenting Tang, Fan Du, Matt W. Mutka, Lionel M. Ni, and Abdol-Hossein Esfahanian, *Supporting global replicated services by a routing-metric-aware DNS*, Proceedings of the 2nd International Workshop on Advanced Issues of E-Commerce and Web-Based Information Systems (WECWIS 2000), June 2000, pp. 67–74.

[8] Maarten van Steen, Philip Homburg, and Andrew S. Tanenbaum, *Globe: A wide-area distributed system*, IEEE Concurrency **7** (1999), no. 1, 70–78.